리페이싱 : 다계층 아키텍처에서 표현 계층의 리팩토링□

이욱진° 박상현 이병정* 김희천** 우치수
서울대학교 전기·컴퓨터공학부, 서울시립대학교 컴퓨터과학부*, 한국방송통신대학교 컴퓨터과학과**
{duri96°, zez4shy, wuchisu}@selab.snu.ac.kr, bjlee@venus.uos.ac.kr*, hckim@knou.ac.kr**

Refacing : Refactoring of Presentation Layer in n-Tier Architecture

Wook-jin Lee° Sanghyun Park, Byungjeong Lee* Heechern Kim** Chisu Wu
School of Computer Science & Engineering, Seoul National University
School of Computer Science, University of Seoul*
Dept. of Computer Science, Korea National Open University**

요 약

리페이싱은 다계층 아키텍처에서 클라이언트가 접하는 표현 계층을 리팩토링하는 것이다. 즉 리페이싱은 표현 계층의 부적절한 구조 및 코드를 찾아내어 적절하게 개선하는 작업이다. 리페이싱은 1) 코드 가독성을 향상시키고, 2) 유지보수를 원활하게 하며, 3) 시스템이나 제품의 업그레이드 과정을 도와준다. 이 논문은 리팩토링을 참고하여, 리페이싱 절차를 제안하고, 실제로 카탈로그 중 하나인 '표현 계층과 업무 처리 계층(Business Logic Layer)의 분리' 카탈로그를 소개한다.

1. 서 론

정보시스템의 규모가 커지면서 다계층 아키텍처는 기술의 종류에 관계없이 중요한 아키텍처로 자리 잡았다. 특히 클라이언트/서버(C/S) 시스템이나 웹 응용 시스템의 개발 과정에서는 다계층 아키텍처를 널리 채택하고 있다. 이것은 개발 생산성 및 유지보수의 용이함을 감안할 때, 필연적인 현상이다. 각 계층을 독립적으로 개발하고, 유지보수 할 수 있기 때문에 더 짧은 기간에 더 적은 비용으로 개발 및 유지보수 할 수 있기 때문이다.

현실적으로 주어진 자원(개발 기간과 비용)만으로 결함이 없고 만족할 만한 시스템을 만들기 힘들기 때문에, 소프트웨어 공학자들은 시스템을 개선하는 많은 절차나 방법론을 제안했다. 대표적으로 객체 기반 기술의 업무 처리 계층(Business Logic Layer)을 개선하기 위한 절차를 정리한 '리팩토링'을 예로 들 수 있다.

리팩토링은 외부 동작을 바꾸지 않으면서 내부 구조를 개선하는 방법으로, 소프트웨어 시스템을 변경하는 프로세스이다. 변경 과정에서 버그가 끼어들 가능성을 최소화면서 코드를 정리하는 정형화된 방법이다.

그런데 클라이언트, 특히 사용자가 직접 맞대는 표현 계층을 개선하기 위한 절차나 방법론은 드물다. 때문에 표현 계층의 개선을 위한 절차를 체계적으로 정리한 '리페이싱(Refacing)'을 제안한다. 그리고 실제로 절차를 정리한 카탈로그를 소개한다.

최신 기술을 기반으로 새롭게 개발하는 시스템은 표현 계층과 다른 계층을 잘 분리하고 있지만, 대부분의 시스템은 과거 기술에 기반하고 있다는 점을 감안하면 리페이싱의 대상은 매우 넓다고 볼 수 있다.

이 논문에서는 웹 응용 시스템의 리페이싱만을 고려한다. 2장에서는 리페이싱의 절차를 정의하고, 3장에서는 리페이싱 카탈로그가 갖춰야 할 요구사항을 기술한다. 4장에서는 리페이싱 카탈로그 중 한 가지를 소개하고, 5장에서는 관련 연구를 검토한다. 끝으로 6장에서 연구의 의의 및 향후 과제를 알아본다.

□ 본 연구는 한국과학재단 목적기초연구(R01-2002-000-00135-0)지원으로 수행되었음.

2. 리페이싱의 절차

리페이싱의 절차는 다음 순서로 표현할 수 있으며, 기본적으로 리팩토링과 비슷하다. 리팩토링과 차이점은 각 항목에서 설명한다.

1) 좋지 않은 징후(Bad Smell)를 발견한다.
2) 테스트 집합을 만든다.
3) 기능은 그대로 둔 채, 구현 방법만 바꾼다.
4) 테스트 한다.

2.1 좋지 않은 징후 (Bad Smell)

무턱대고 리페이싱을 할 필요는 없다. 리페이싱에서도 리팩토링과 마찬가지로 좋지 않은 징후가 있는 곳을 먼저 발견한다. 현재 정리한 징후는 다음과 같다.

- 업무 처리 기능이 표현 계층에 위치한 경우
- 자바 스크립트와 같은 클라이언트 스크립트가 HTML 코드와 혼재되어 있는 경우
- 항해 정보(Navigation Information)가 표현 계층에 혼재되어 있는 경우

이런 징후를 발견하면, 카탈로그를 참조[1]하여 어떤 리페이싱 절차를 따를 것인지 결정한다. 예를 들어 위의 3가지 징후는 각각 '표현 계층과 업무 처리 계층의 분리', '자주 사용하는 클라이언트 스크립트 추출', '페이지와 항해 정보의 분리' 카탈로그를 참조할 수 있다.

2.2 테스트 집합

리페이싱 절차를 따라 변경한 다음, 변경 영향(Change Effect)가 있는지 확인할 수 있는 테스트 집합이 필요하다. 리팩토링과 달리 JUnit 테스트 수트를 사용할 수 없기 때문에 테스트 절차가 더 번거로울 수 있다. 그렇지만 Apache Jakarta 프로젝트의 JMeter를 사용하면[2], 파라미터 기반의 웹서버 - 클라이언트 통신을 재현하는 등 많은 도움을 받을 수 있다.

2.3 구현 방법의 개선

구현 방법의 개선은 카탈로그의 세부 절차를 따른다. 중요한 것은 리팩토링과 마찬가지로 각각의 세부 개선 절차는 eXtreme Programming (XP) 방식을 따른다는 점이다.

2.4 테스트

'2.2'에서 준비한 테스트 집합을 이용하여 테스트한다. 기능이 변화하지 않았는지 확인한다.

3. 리페이싱 카달로그 요구사항

리페이싱 카달로그는 리팩토링 카탈로그와 같이 '요약 - 동기 - 절차 - 예제' 순서로 정리한다. 각각의 단계가 갖춰야 할 요구사항은 다음과 같다.

- 요약 : UML 형태로 표현한다. 필요한 곳에 스테레오 타입을 명시하여 명확히 의사 전달을 할 수 있도록 한다. [3]
- 동기 : 어떤 상황에 카탈로그를 적용해야 하는 지 설명한다. 현재는 카탈로그의 수가 많지 않아 카탈로그를 잘못 참조할 가능성이 적지만, 카탈로그의 가지 수가 많아지면 어떤 경우에 적용하지 말아야 할지도 설명한다.
- 절차 : 작은 단위로 상세히 절차를 설명한다.
- 예제 : 절차를 따라 실제 적용한 예제를 설명한다.

본 논문에서는 분량 제한을 감안하여 '표현 계층과 업무 처리 계층(Business Logic Layer)의 분리' 한 가지만을 설명한다. 보다 많은 카달로그는 리페이싱 사이트[1]를 참조한다.

4. 카달로그 : 표현 계층과 업무 처리 계층의 분리

4.1 요약

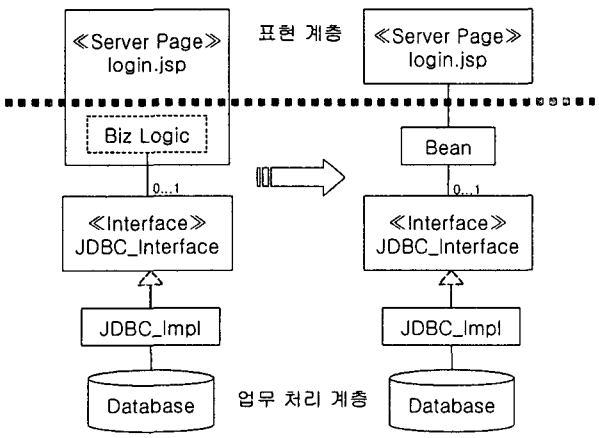

그림 1 '업무 처리 계층과 분리' 카탈로그 요약

표현 계층의 서버 페이지가 업무 처리 기능을 가지고 있는 것은 유지 보수 측면에서 좋지 않은 경우가 많다. 특히 서버 페이지가 직접 데이터베이스와 통신하는 경우도 적지 않게 발견할 수 있다. 일반적으로 서버 페이지가 포함하고 있는 업무 처리 기능을 별도의 클래스로 추출하여 계층 사이의 분리를 명확히 하는 것이 좋다.

4.2 동기

많은 개발자들이 개발 생산성을 이유로 표현 계층의 서버 페이지에 업무 처리 기능을 구현한다. 이것은 작은 규모의 시스템에서 도움이 될 수 있지만, 큰 규모의 시스템에서는 코드의 가독성을 떨어뜨리고, 유지보수를 어렵게 하는 등 많은 단점을 가지고 있다. 따라서 그림 1과 같이 별도의 클래스로 추출한다.

4.3 절차

1) 표현 계층의 서버 페이지의 테스트 수트를 만든다.
2) 서버 페이지가 가지고 있는 업무 처리 부분의 테스트 수트를 만든다.
3) 서버 페이지의 업무 처리 부분을 추출하여 업무 처리 클래스를 만든다.
4) '2)'의 테스트 수트를 이용하여 업무 처리 클래스를 테스트한다.
5) useBean을 사용하여 표현 계층의 서버 페이지가 업무 처리 클래스를 사용하도록 바꾼다.
6) '1)'의 테스트 수트를 이용하여 테스트 한다.

4.4 예제

변경 전의 코드는 그림 2와 같다.

```
<%@page language="java" %>
<%@page contentType="text/html;charset=EUC-KR" %>
<script LANGUAGE="JavaScript">
<!--
<%
  int n_docType = 0;
  n_docType = Integer.parseInt(request.getParameter("DT"));
  String s_CodeName = "", s_ELName = "";

  switch( n_docType ){
    case 1 :
      s_CodeName = "RuleCode";
      s_ELName = "RuleData";
    break;
    case 2 :
    case 5 :
      s_CodeName = "SubCategoryCode";
      s_ELName = "SubCategory";
    break;
    case 3 :
      s_CodeName = "ChapterCode";
      s_ELName = "ChapterData";
    break;
    case 4 :
      s_CodeName = "SubCategoryCode";
      s_ELName = "SiBang-Data";
    break;
  }
%>
  function goTo(obj, s_CodeName){
    var d = obj.options[obj.selectedIndex].value;
    window.location.href                                    =
"http://10.134.6.61/app/servlet/SB_ViewQuery?ELN=<%=s_ELNa
me%>&<%=s_CodeName%>="+ s_CodeName + "&CN=" + d;
  }
//-->
</script>
```

그림 2 '업무 처리 계층과 분리' 카탈로그 적용 전

s_CodeName, s_ELName 두 변수의 값을 n_docType 변수에 따라 결정하는 업무 처리 부분을 가지고 있다. 절차를 따라 업무 처리 부분을 클래스로 만든 코드는 그림 3과 같다. 이 클래스를 이용하여 개선한 서버 페이지의 코드는 그림 4와 같다.

```
package kr.ac.snu.selab.refacing;
public class Bean {

private String codeName = "", elementName = "";
private int docType = 0;

public void setDocType(int n_docType) {
        docType = n_docType;
        switch (n_docType) {
        case 1:
                codeName = "RuleCode";
                elementName = "RuleData";
                break;
        case 2:
        case 5:
                codeName = "SubCategoryCode";
                elementName = "SubCategory";
                break;
        case 3:
                codeName = "ChapterCode";
                elementName = "ChapterData";
                break;
        case 4:
                codeName = "SubCategoryCode";
                elementName = "SiBang-Data";
                break;
        }
}
public String getCodeName() { return codeName; }
public String getElementName() { return elementName; }
public int getDocType() { return docType; }
}
```

그림 3 업무 처리 부분만을 떼어 내어 만든 클래스

```
<%@page language="java" %>
<%@page contentType="text/html;charset=EUC-KR" %>
<script LANGUAGE="JavaScript">
<!--
<jsp:useBean id="bean" class="kr.ac.snu.selab.refacing.Bean"
/>
<%
  int n_docType = 0;
  n_docType = Integer.parseInt(request.getParameter("DT"));
  bean.setDocType(n_docType);
%>
  function goTo(obj, s_CodeName){
    var d = obj.options[obj.selectedIndex].value;
    window.location.href =
"http://10.134.6.61/app/servlet/SB_ViewQuery?ELN=<%=bean.g
etElementName()%>&<%=bean.getCodeName()%>="+
s_CodeName + "&CN=" + d;
  }
//-->
</script>
```

그림 4 '업무 처리 계층과 분리' 카탈로그 적용 후

5. 관련 연구

리페이싱이라는 용어는 현재 IBM사에서 자사 메인프레임 제품의 사용자 인터페이스를 웹 브라우저 기반의 인터페이스로 변환하는 작업을 지칭하는 용도로 사용하고 있다.[4,5] 이외에 소프트웨어 공학 분야에서 이 용어를 주목할 정도로 사용한 경우는 발견하지 못했다.

리팩토링은 전술한 바와 같이 외부 동작을 바꾸지 않으면서 내부 구조를 개선하는 방법으로, 소프트웨어 시스템을 변경하는 프로세스이다.[6] 리페이싱은 리팩토링의 개념, 절차 등을 참고했다.

표현 계층의 다이어그램 표현은 UML 형태를 따랐다. 따라서 페이지 표현, 항해 정보 표현 등은 모두 스테레오 타입을 이용하여 표현했다.

6. 결 론

개발 프로젝트 기간 중, 구현 기간은 점점 짧아지고 있다. 여기에 더하여 프로젝트 수행 회사의 지속성 문제, 프로젝트 수행 개발자의 이직 문제 등 현장의 개발 환경은 나날히 악화되고 있다. 이런 점을 감안할 때, 바람직하지 않은 표현 계층을 개선하는 리페이싱의 중요성이 커지고 있다.

앞으로 두 가지 방향으로 이 연구를 더 깊이 수행해야 한다. 우선 JCP의 자바서버 페이스(JavaServer Faces)나 마이크로소프트의 웹폼 기술 등도 반영할 수 있도록 리페이싱 절차를 심화해야 한다. 그리고 역공학적 접근을 이용하여 좋지 않은 징후를 더 쉽게 발견하고, 고칠 수 있도록 심화해야 한다.

또한 현재의 카탈로그를 더 확장하여 보다 풍부한 카탈로그를 갖추는 것에도 노력해야 한다.

참고 문헌

1. 이욱진, Refacing, http://selab.snu.ac.kr/~duri96/refacing.html, 2004.
2. Apache Jakarta Project, JMeter, http://jakarta.apache.org/jmeter/, 2004.
3. OMG, UML (Unified Modeling Language), http://www.uml.org/, 2004.
4. IBM, WebSphere Development Studio Client for iSeries / Features and benefits http://www-306.ibm.com/software/awdtools/wdt400/about/, 2004.
5. IBM, IBM iSeries Education (Application Development) / Refacing an iSeries Green Screen Application using WebSphere Development Studio Client V5.0 - Advanced Customization LAB http://www-1.ibm.com/servers/enable/site/education/abstracts/webclientadv_abs.html, 2004.
6. Martin Fowler, Refactoring, *Addison-Wesley Professional*, 1999.